Real-time synchronization of independently controlled phasors

Lonce Wyse

National University of Singapore, Singapore
lwyse@zwhome.org

Abstract

Phasors have a wide variety of uses in music and sound synthesis. At audio signal rates, phasors can be used as indexing functions for table oscillators. At control rates, phasors can be used as clocks to control tempo and trigger events. When simultaneous control over multiple phasors or oscillators is desired, there arises a conflict between the need to control their individual frequencies, and the need to coordinate the phase relationships between the different phasors. The problem is particularly acute in real-time systems where a flexible mechanism for negotiating between local and global control is necessary. This paper presents a spline technique for quasi-independent control over phasor frequency and phase, discusses the objects and interfaces in an implementation, and presents several examples to illustrate the utility of the system for signal and event generation, and for synchronization in real-time networked music environments.

1 Introduction

The objective of the system described herein is to provide a method for synchronizing independent phasor-based musical or signal processes at specific points of time in a real-time performance environment. Coordinating independent musical voices at specific points in time has typically been addressed using the idea of "time maps" (Jaffe 1985, Honing 2001) or "time warping" (Dannenberg, 1997). The idea is to use time warping functions that map linear time *t* to a "warped time" time *t'*:

$$f(t) \rightarrow t' \quad . \tag{1}$$

Voices are synchronized where their warping functions intersect (Figure 1). A recent implementation of multi-voice "tempo cannons" using time maps is discussed in Collins (2003).

One aspect that differentiates the current work from previous time map formulations is that the focus is on the cyclic behavior of phasors. For the purposes of this paper, we consider a phasor to be a function of time *t* that maps periodically into the unit interval at a rate of *freq* cycles per second:

$$\phi(freq, t) \rightarrow [0,1] \tag{2}$$

We will also consider the "unwrapped" phase which is the total number of periods executed by the phasor over some duration *t* (during which the frequency may be changing continuously). Taking the modulus 1 of the unwrapped phase yields the "principle phase" in the unit interval.

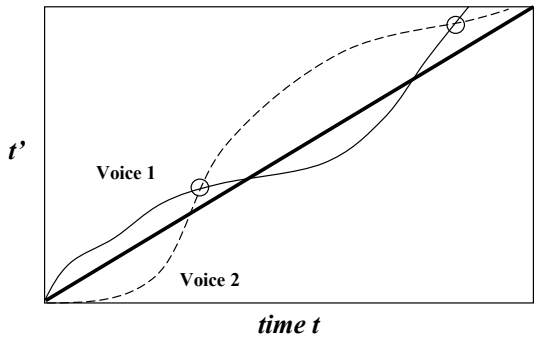

Figure 1. Time maps for two voices that intersect (are "synchronized") at two points during the interval shown. The diagonal line represents a linear map with no time warping .

When synchronizing phasors, it is only relative phase that it is important, not the number of cycles accumulated from some reference time (e.g. the beginning of a piece of music). For example, consider two improvising musicians, each with control over periodic musical sound-generating algorithms ("models") that are independently parameterized in real-time. One of the model parameters controlled by each musician is the frequency of a phasor that generates events at specific phases in its cycle (Figure 2). To create synchrony between the two independently controlled voices, a time warping mechanism would only need to consider a localized window of time rather than the entire history of the piece.

We commonly think of a phasor as having two attributes, frequency and phase. In the musical scenario described above, both the frequency of the event generating processes and the phase relationship between the two would be clearly

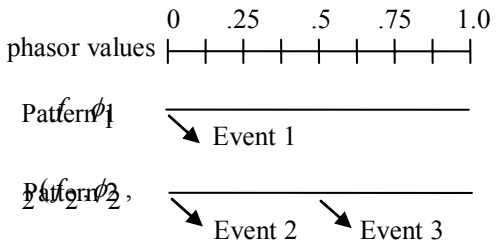

Figure 2. Two independently controlled cyclic patterns with events triggered at specific phasor values. Because the patterns are cyclic, there is no single reference time for synchronization. Instead, a reference is chosen in real-time and warping depends only on phase.

perceptible. Of course, the two attributes are not independent since frequency is just the rate at which phase changes. The only parameter the musicians have available for controlling the synchrony between their individual beat patterns is the frequency of their respective phasors. Skilled musicians would generally be able to maintain any particular phase relationship by continually adjusting their respective frequency parameters (at the expense of maintaining the exact frequency target). However, a more difficult task would be for the two musicians to synchronize both the frequency and phase at a specific forward point in time that was itself determined in real-time. The task is difficult precisely because there is no independent control of frequency and phase. Even if the musicians were computers, how can a real-time system provide the sense that both frequency and phase are under independent control so that specific frequency and phase configurations can be achieved at specific points in time? This is the task to which the technique described in this paper is addressed.

2 Real-time time warping

One way to provide quasi-independent real-time control over both frequency and phase in real time is to expose both attributes as parameters that control "target values" and then permit the system a certain (specifiable) duration of time in which to achieve the targets. The system would have the freedom to automatically manipulate the frequency over time so that the desired phase is reached at the exact point in time specified by the duration. During the transition interval, the frequency will have to be different from the target frequency, and in general, continuously changing. However, if the system is designed so that the transition time is short, and the frequency is constrained to deviate as little as possible from its target, then the sense of real-time frequency control need not be lost as phase control is gained.

The warping functions that we will consider will map time to the unwrapped phase of an oscillator. Thus a map representing a phasor of constant frequency would appear as a line from the origin with a slope equal to the frequency. "Warping" amounts to deviating from the linear map to achieve specific phases and frequencies at the endpoints.

In musical applications, at both signal and event control rates, a smoothness condition would typically be applied at the beginning and the end of the transition so that there are no sudden changes in frequency (or "tempo" if the phasors are triggering events). That is, we would like to specify the frequency of the phasor at the endpoints of the transition, and would most often choose the starting value to be equal to the frequency at the point when the transition was called for, and the endpoint to be equal to whatever frequency we want the phasor to continue with following the transition. Since the mapped value represents phase, and the derivative phase is the frequency, this amounts to the desire to specify the derivative of the mapping function at the endpoints.

Precomputed mapping functions are not possible for this task because of the requirement of being able to specify the endpoint derivatives at run time. What is needed is a simple and efficient method for computing the time-to-phase mapping functions on the fly that permit us the desired specifications of transition duration, start and end point derivatives, the number of total phasor cycles, and some way to control the maximum deviation of the warping. Clamped cubic splines fit the bill precisely.

Splines are a technique for interpolation between known values of a function f at a sequence of points: f(x1), f(x2) …. f(xn), with polynomials between each pair of points. A cubic spline is constructed of third-order polynomials between the specified function values. A clamped cubic spline uses specified derivate values at the endpoints. The qualities of clamped cubic splines that are useful in the musical time map context are:

- they are smooth in the first derivative, and continuous in this second at the endpoints as well as all points in between. Musically, this means there are no abrupt frequency/tempo changes,
- the function achieves the specified values (phasor values) exactly at tabulated points,
- the "clamped" conditions that give this type of spline its name are the slopes of the function at the end points, which correspond to the frequency of the phasors at the beginning and end of the transition period,
- the computational effort for solving the system to create the spline function is linear in the number of tabulated points.

Details on how splines are computed can be found in standard texts (cf. Press et al., 1988). Once the spline map has been computed following a real-time parameter change, then the warped function value is available continuously at any x value between the specified endpoints.

3 Phase tracking

"Tracking" is the term we use for the process of adjusting phasor frequency continuously over time to reach specific frequency and phase values. A phasor with tracking capabilities has been built into a pure Java library and real-time synthesis system described in Wyse (2003). The phasor object and application programmer interfaces (API's) are described briefly below because they illustrate some of the issues that arise in synchronizing phasors in real time.

Phasor objects are initialized with a sample rate, *sr,* and controlled with a frequency parameter, *freq*. After initialization, the phasor is "sampled" by calling the Phasor "tick()" method which will advance the phase by *freq/sr* and return the new phase value.

To initiate the synchronization transition, phasors objects have a family of trackX() methods. When they are called, a spline function is created to the specification supplied in the trackX() argument list. The phasor goes into a "tracking mode" that lasts for the duration specified to achieve the desired phase and frequency. During that time, the advance of the phase value in response to the "tick" method is determined not by the normal phasor frequency parameter, but is set on a sample-by-sample basis by the spline function values until it has achieved the desired end phase and frequency after exactly the specified duration.

Different trackX() methods allow for different ways of specifying the desired behavior during the spline interpolation interval:

- trackCycles(*numcycles, targetfreq, duration*) – rotates through an exact (real-valued) number of cycles over the specified duration and clamps the slopes at the spline endpoints to the current phasor frequency at the beginning, and the *targetfreq* argument at the end. This method is typically called by other trackX() methods, rather than sound model programmers because it requires the caller to know current phases, and compute the desired number of cycles necessary to arrive at the implicitly desired phase that will create synchrony across different phasors.
- trackPhase(*targetphase, targetfreq, duration*) – this method affords a more user-friendly argument list and internally computes the actual number of cycles the phasor must execute over the duration to arrive at the desired target phase and target frequency. Of course, the number of cycles to execute is not defined without further conditions because for any real-valued number of cycles that satisfies the argument conditions, adding an integer number of cycles will also satisfy the conditions. The actual number of cycles is computed as the average of the starting and ending frequencies times the duration, and is then adjusted the minimal amount to meet the phase requirement. Note that the phase adjustment might be positive or negative; the one of smaller absolute value is chosen for minimum frequency perturbation.
- trackPhase(*targetphase, targetfreq, duration, direction*) – the same as above, except the adjustment for phase is forced to go in either the positive or negative direction. This turns out to be particularly important when the total number of cycles is small and carried out over control/event time scales.
- trackPhase(*ref_freq, ref_phase, target_phase_rel, targetfreq, duration)* – another convenience method that allows phase synchronization to a (fixed-frequency) reference phasor. This interface permits a conceptualization of the synchronization process in terms of musical beats rather than clock time.

3.1 Traveling Backwards

Jaffe (1985) disallowed time maps with negative derivates – those that represent traveling backwards through the score. In our case, this corresponds to running a phasor with negative phase increments. Mappings considered here are from real-time to phase values, and never the other way around. The need for representing or computing inverse maps does not arise as it does when computing maps from score time (Dannenberg, 1997), so there is no other reason than a composer's intentions for preventing negative derivatives. However, if we allow negative derivatives to be computed in a real-time system, then some extra care is required, particularly at the slow time scales that typically control events. The danger is that with parameters changing in real-time, continuously recomputed time maps that permit negative derivatives can result in the process repeatedly "scratching" over a particular phase region and rapidly retriggering any events that are hooked to phases in the region. With cubic splines, this situation arises quite commonly in practice when slowing down an oscillator where the phase advancement across the endpoints is small and the derivative (frequency of the phasor) is the same at both endpoints (Figure 3). The minimal order cubic spline tends to create the characteristic S shape with a negative derivative between the two extrema.

There are two ways that the problem can be addressed. One is to increase the order of the spline so that there are

more segments. This adds to the computation since the spline needs to be recomputed for every parameter update (which in the situation described is frequent). Notice that in cases where the negative derivative happens over phase values that are not associated with events, then the shape of the map function is not manifested sonically at all. A

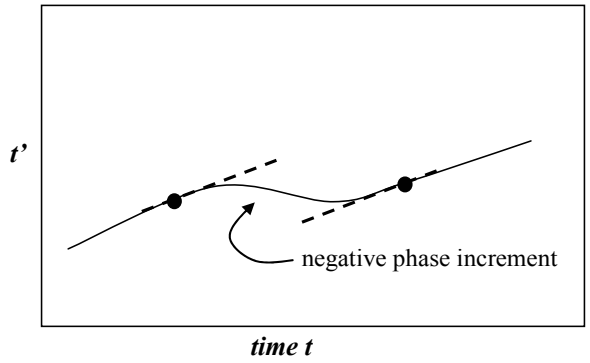

Figure 3. When the phase advancement over the duration of the spline is small, the cubic s-shaped interpolator can generate a negative slope causing the phasor to to run in reverse. If there is any sounds or events tied to the phase values in this range, this leads to "scratching" repeatedly over them .

straight-forward way to avoid the dangers of changing the direction of phase increments is to simply use the low-order spline with its negative derivative, but fix the value of the event phasor during these periods until the spline tick() method returns phases that represent positive increments again. In this situation, this strategy prevents scratching as effectively as increasing the order of the spline, but without the additional computational burden.

4 Musical Possibilities

Wave terrain synthesis (cf. Roads, 1996) is one example of a signal-rate application that can benefit from the ability to manipulate phase and frequency quasi-independently. Wave terrain synthesis uses a 2D table as a transfer function, and a 2D indexing function. A common indexing method is a Lissajou function, a curve generated by two sinusoidal oscillators; one that determines the x-coordinate, the other that determines the y-coordinate used for reading an interpolated value from the 2D wavetable (Figure 4).

We can think of the indexing oscillators as sinusoidal functions of phasor values in order to use the concepts and implementation described above. If the phasor-determined frequencies of the two indexing oscillators are not related by an integer multiple, then the resulting Lissajou figure appears to be a 2-D projection of a contour rotating in 3 dimensions. Specifying phases at particular points in time only shifts the resulting sound in time. However, if the frequency of one oscillator is an integer multiple of that of the other, then a stable 2-D path is traced periodically. In

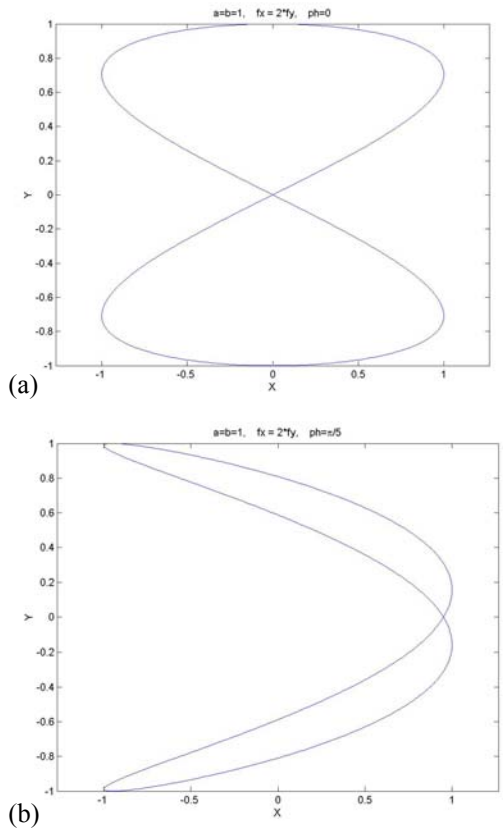

Figure 4. Two Lissajou figures each with frequency(x) = 2*frequency(y) a) with a 0 phase difference between the two sinusoidal indexing functions at time t=0, and b) with a pi/5 phase difference between the two indexing functions at time t=0.

this case, the shape of the Lissajou figure, and thus the elements being read from the 2D wavetable, depends significantly on the phase difference between the two indexing oscillators (Figure 4a,b), and the "natural" parameter for controlling the system becomes phase rather than frequency.

It may be that short transition times and minimal frequency deviation is not the desired use for this synchronization mechanism. Long slow transition times between synchronization points can be put to wonderful musical effect as demonstrated by, for example, Colin Nancarrow's Player Piano Study #21. This piece is for two voices with a cyclic event pattern, and uses the entire 3 minute duration of the piece to make the transition from the event phase synchronization at the beginning to that at the end.

These phasor synchronization techniques can be used musically in a way analogous to "keyframes" in animation. Consider multiple voices, each generating a periodic pattern of events; a simple example would be a set of church bells each with their own pitch and period. Keyframes in this context are specific patterns across voices such as different

arpeggios. A performer could then, in real time, choose one of the predefined keyframes, and set a time interval during which the individual voices would all have their periods minimally adjusted so that after the specified time, the keyframe event pattern would be produced. Spline maps defined only by two endpoint values and slopes produce an "ease in, ease out" effect that is frequently useful musically, and is also analogous to animation keyframing techniques.

Another important context for the real-time synchronization capabilities described above is network communications. With fast internet communications capable of carrying live video and audio data, the prospect of musicians who are physically located around the globe to jam together becomes feasible. However, dividing half the circumference of the earth (20K kilometers) by the speed of light (300K km/s) we see that physics imposes a minimum delay of 67 ms on audio communication at this distance. In practice, network switching and buffering typically create delays of longer than this. If the network is carrying only low-bandwidth parameters for the remote control of synthesis (rather than audio signals), then jitter in the timing also becomes an issue. These factors would seem to make tight synchronization between distant musicians impossible.

One solution takes advantage of new relationships between performance gesture and sound events that are being explored in contemporary music whether networked or not. If the one-to-one mapping between gesture and musical event that characterizes traditional instrument performance is abandoned in favor of mapping gesture to less time-sensitive parameters or second order timing parameters, then satisfying remote real-time musical collaborations can still be supported. If, for example, tempo, syncopation, or cross-voice synchronization were the parameters under control rather than individual events, then absolutely precise timing between multiple event streams controlled by remote performers is possible under any delay and jitter conditions. Of course, there would still be limits on precisely *when* synchronization would occur but not on how precisely the synchronization could be achieved between the remotely and locally controlled voices. Performers on both sides of the network divide could be sure of the results of their performance actions, and that they were having the same audio experience as that of their distant counterpart.

5 Summary

A method of giving performers a sense of independent control in real-time over both the frequency and relative phase across multiple phasor-driven processes was presented. The method is akin to classic time maps discussed in earlier literature, and is based on a clamped spline interpolation that is specified with musically meaningful parameters. An implementation and interfaces were developed in a musical context, and examples at both the signal and the event control rate show the utility of the method in a variety of realistic real-time musical environments.

References

Collins, N. 2003. A microtonal tempo canon generator after Nancarrow and Jaffe. In *Proceedings of the International Computer Music Conference.* Singapore: International Computer Music Association.

Dannenberg, R. B. 1997. Abstract Time Warping of Compound Events and Signals. *Computer Music Journal* 21(3), 61–70.

Jaffe, D. 1985. Ensemble Timing in Computer Music. *Computer Music Journal* 9(4), 38-48.

Honing, H. 2001. From time to time: The representation of timing and tempo. Computer Music Journal, 35(3), 50-61.

Press, W.H., B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, 1992. *Numerical Recipes in C*, 2nd edition, Cambridge University Press.

Roads, C. 1996 *The Computer Music Tutorial.* Cambridge, Mass.: MIT Press, 163-167.

Wyse, L. 2003. A sound Modeling and Synthesis System Designed for Maximum Usability. In *Proceedings of the International Computer Music Conference,* 447-451. Singapore: International Computer Music Association.